RasPi

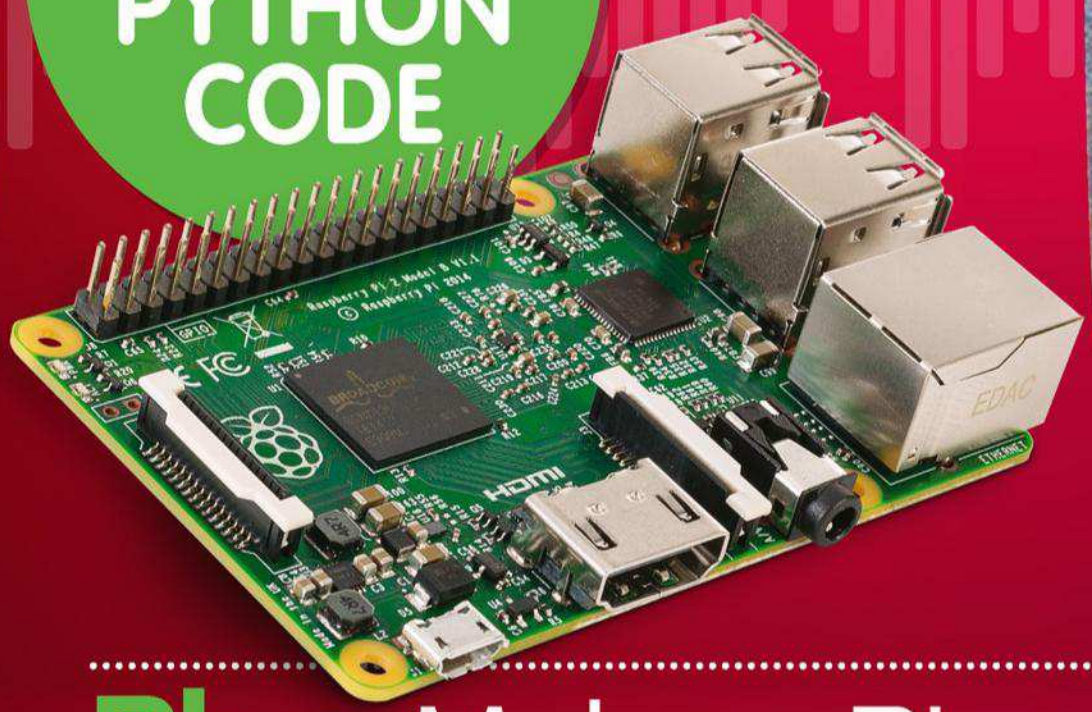
DESIGN
BUILD
CODE

48

Get hands-on with your Raspberry Pi

BUILD A PI VOICE ASSISTANT

+ SPEED UP PYTHON CODE



Plus Make a PI weather station

Welcome

Voice assistants are all the rage at the moment, what with Microsoft's Cortana, Apple's Siri, Google's Home and Amazon's Alexa all entering the market. This issue, Raspi magazine is going to show you how to create a fantastic open source equivalent that has Raspbery Pi at it's heart. One of the great benefits of this is you'll know exactly what data is being collected and by whom, so swipe left and have a go yourself. All you'll need is a Raspberry Pi, a USB microphone and some speakers. Other highlights this issue include building a custom weather station and speeding up your Python code with Numba.

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

Learn to make and code gadgets with Raspberry Pi

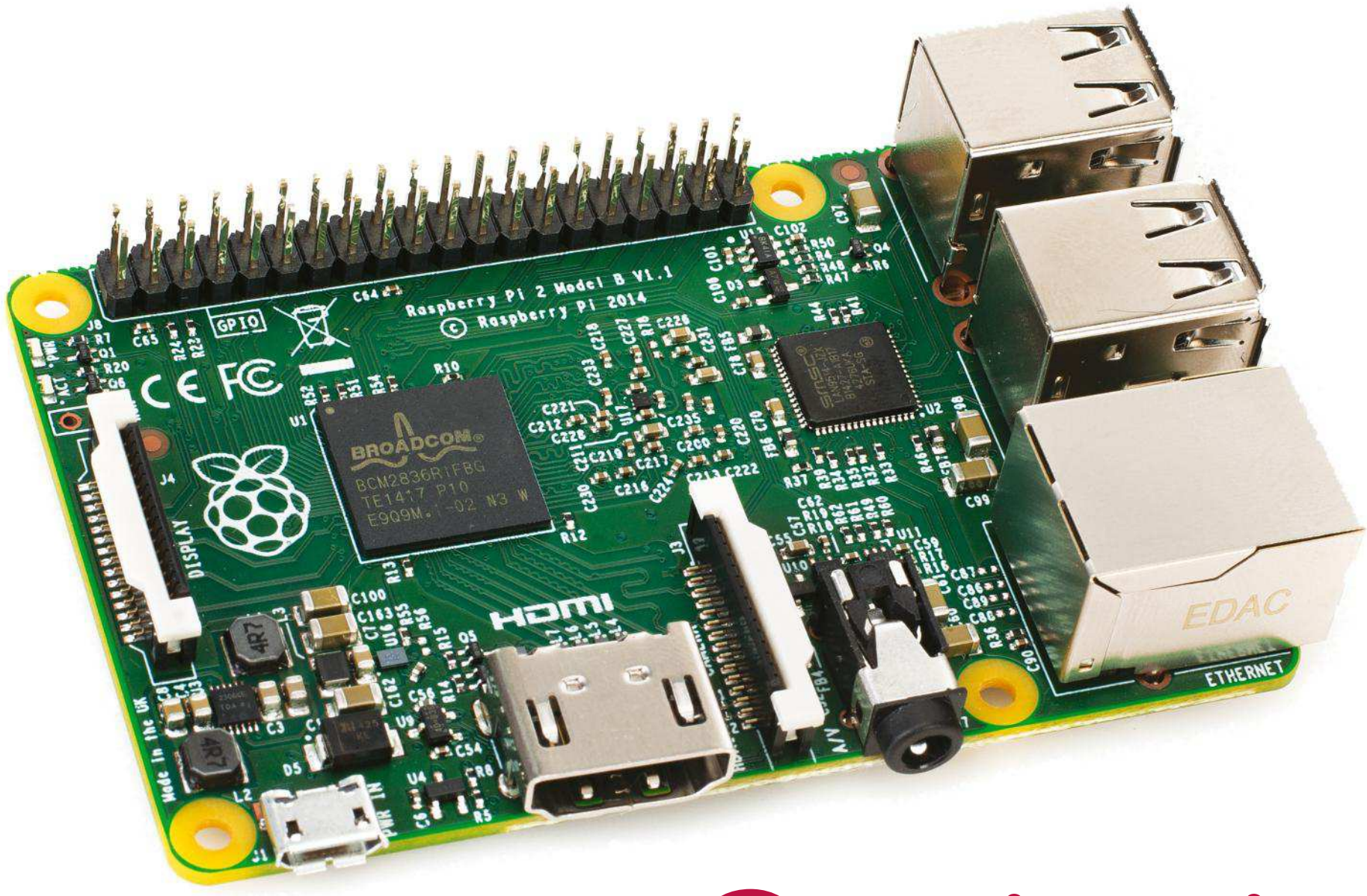
Editor

From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | linuxuser@futurenet.com



Contents

Make an open source voice assistant with Mycroft

Forget Cortana, Alexa, Google Home and Siri – we're going open source and creating our own voice assistant

Voice assistants are all the rage at the moment, what with Microsoft's Cortana, Apple's Siri, Google's Home and Amazon's Alexa all entering the market. Users are becoming more comfortable talking to a device and receiving audible instructions, in a way that's not too dissimilar from the computer in the Star Trek franchise. However, with current concerns regarding privacy, it's important to know what data is collected, where it's going, and who could potentially be eavesdropping on your conversations.

We don't mean to sound paranoid, but if you've got an open mic in your environment it's pretty important to know where any data might be heading; some of the larger corporations collect information about users to better target advertisements towards them. That's why users are turning to open source alternatives. This issue we're having a go at building one of these units for ourselves, armed only with a Raspberry Pi, a USB microphone and some speakers.

THE PROJECT ESSENTIALS

Mycroft
https://mycroft.ai/get-mycroft

Raspberry Pi 3

microSD card, 8GB or larger

USB microphone

Speakers

Etcher
https://etcher.io

01 Download and flash

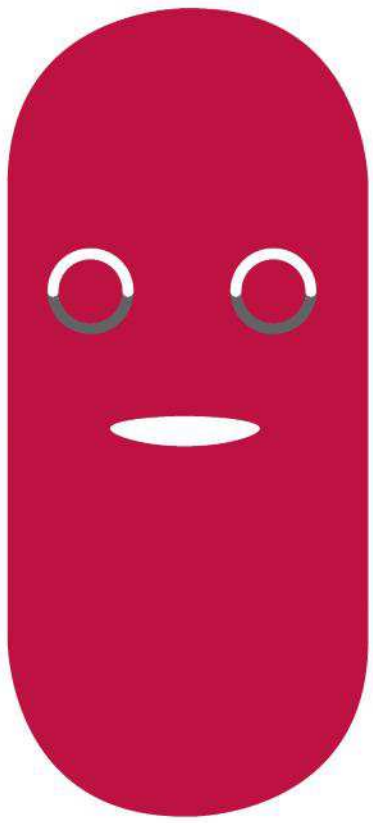
There are Linux (Arch, Fedora and Ubuntu/Debian) and Android versions of Mycroft available, but for this tutorial we're sticking with the Raspberry Pi flavour. We recommend you use a Pi 3.

Download the latest version of Picroft from the link in our Resources section, as well as Etcher. Other than any potential 'Skills' you want to add later on and that should be all you need to download for this tutorial. Etcher is an imaging program which we'll use to burn or flash the downloaded Picroft image to an SD card. Plug your microSD card into your computer, launch Etcher and select the Picroft image. Then flash it!

02 Set it up

Plug your microSD card back into your Raspberry Pi and connect it to a power source. The easiest way to get everything working is to connect your Pi to the local network via the Ethernet port. If you do need to use Wi-Fi, look out for an SSID called MYCROFT; the default password is 12345678.

Once everything is connected, you'll want to either plug in a monitor and keyboard, or connect via SSH to do this headlessly. Whether via Ethernet or Wi-Fi, once your device is connected you'll need to visit http://home.mycroft.ai to start the setup process. You'll need to sign in with Google, Facebook or GitHub, or create a new Mycroft account; given that part of the reason for this project is to protect your data from being shared with big corporations, the latter might be advisable!



03 Find your Raspberry Pi

To date, all Raspberry Pi devices start with a MAC address of B8:27:EB, so we can use this to scan our network

for the Pi, if we don't have a monitor/keyboard to connect to it. You could use nmap, for example:

```
sudo nmap -sP 192.168.1.0/24 | awk
'/^Nmap/{ip=$NF}/B8:27:EB/{print ip}'
```

You can also use arp:

```
$ arp -na | grep -i b8:27:eb
```

If your home network is not on the 192.168.1.* subnet, change the command line accordingly.

04 Connect to Picroft

SSH into your Picroft and you'll be taken straight into the Mycroft CLI screen. Usually, this is quite useful, but while we get things set up we want to exit that screen using Ctrl+C to reach a normal command prompt. Here you'll want to do some basic setting-up. First, change the password: type passwd and follow the prompts. Then change the Wi-Fi network settings:

```
sudo nano /etc/wpa_supplicant/wpa_
supplicant.conf
```

Change the network name and/or password in this config file, then press Ctrl+X to exit and save the file. Then type sudo reboot to reload your Picroft.

05 Add your device to Mycroft.ai

Back in your browser at home.mycroft.ai, click 'Add Device' and you'll be asked for a name and location for your Picroft device. You'll also be asked for a registration

"You'll need to sign in with Google, Facebook or GitHub... The latter might be advisable!"

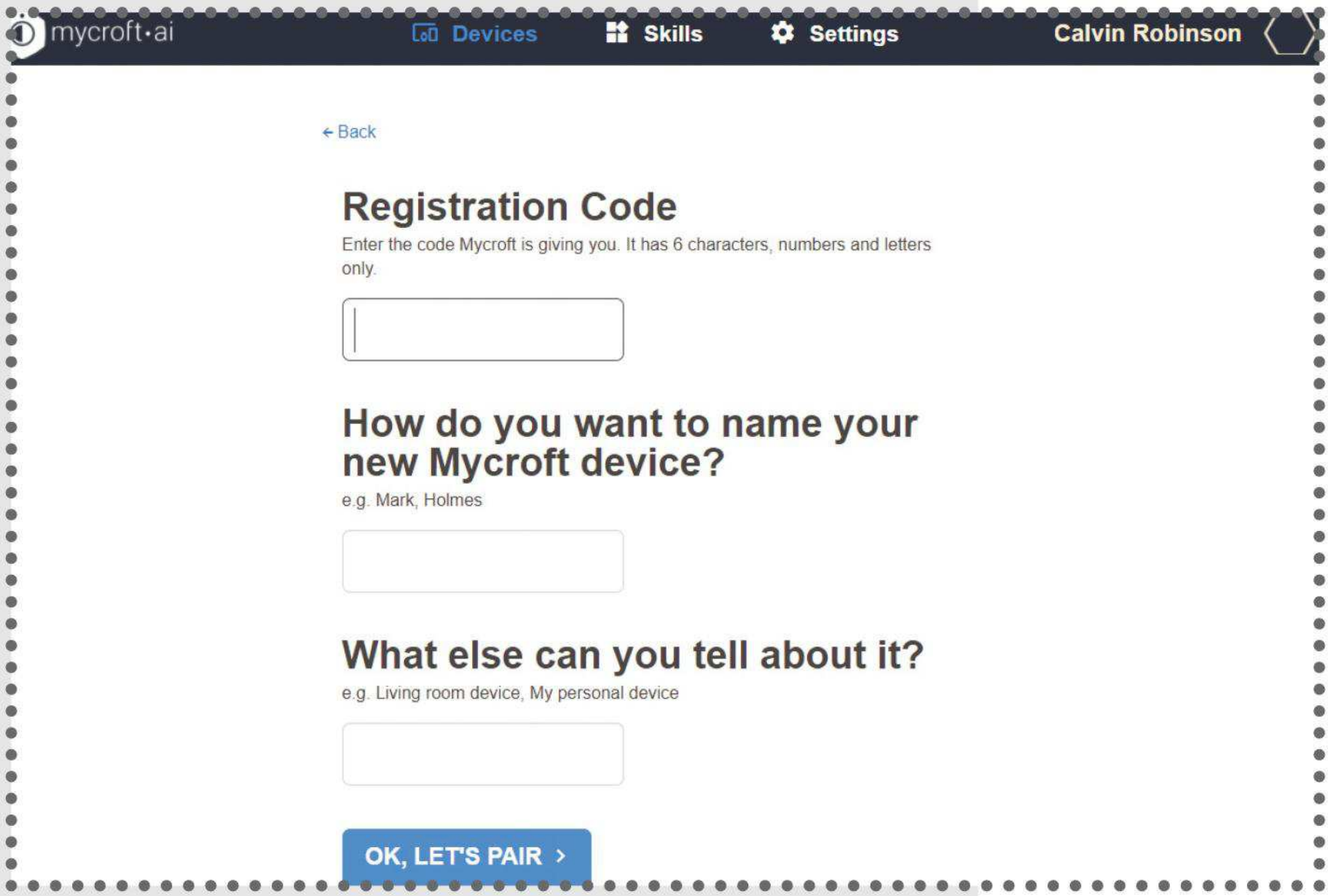
code; if you turn on the speakers connected to your device you'll notice the Picroft is reading this out to you already, until the device is paired up.

Now that we've paired with Mycroft.ai, we can go to the Settings menu where you can select a male or female voice, the style of measurement units you want to use, and your preferred time/date formats.

If you're concerned about privacy, you may want to keep the Open Dataset box unticked. Keep in mind, though, that selecting this option is a good way of contributing useful data to the open source project and thus improving the performance of Mycroft in the future, assuming your voice assistant isn't in a particularly confidential environment.

Mycroft, send help

There's an active community on GitHub ready to help with help requests, which will come in handy as Picroft spits out quite a few Python 2.7 errors when Skills refuse to load properly…



06 Advanced settings

In Advanced Settings, we can really begin to personalise our Mycroft experience. There are a number of pre-programmed wake words, but you can set your own custom version – perhaps 'Computer' a la Star Trek, or maybe 'Butler' if you're feeling particularly bourgeoisie.

You'll need to set the phonetic version of your wake word too, so the device understands what it's listening out for. An example would be 'HH EY . B AH T L ER .' for "Hey Butler". You'll probably want to include some kind of exclamation or greeting before your wake word to avoid confusing the Picroft. This is almost certainly why "Hey Google" or "Okay Google" are used on Google Home, rather than just "Google"; it's to avoid said devices picking up on random conversations, something which happened quite a lot in our testing.

You can also switch the text-to-speech engine from Mycroft's Minic engine to Google's own. This will change the voice you hear to that of Google Home, which is arguably much smoother.

07 Using the Picroft CLI

Now that everything is set up, you should have a basic voice assistant raring to go. Call out the wake word and issue a few commands to get started – Mycroft understands all these examples by default:

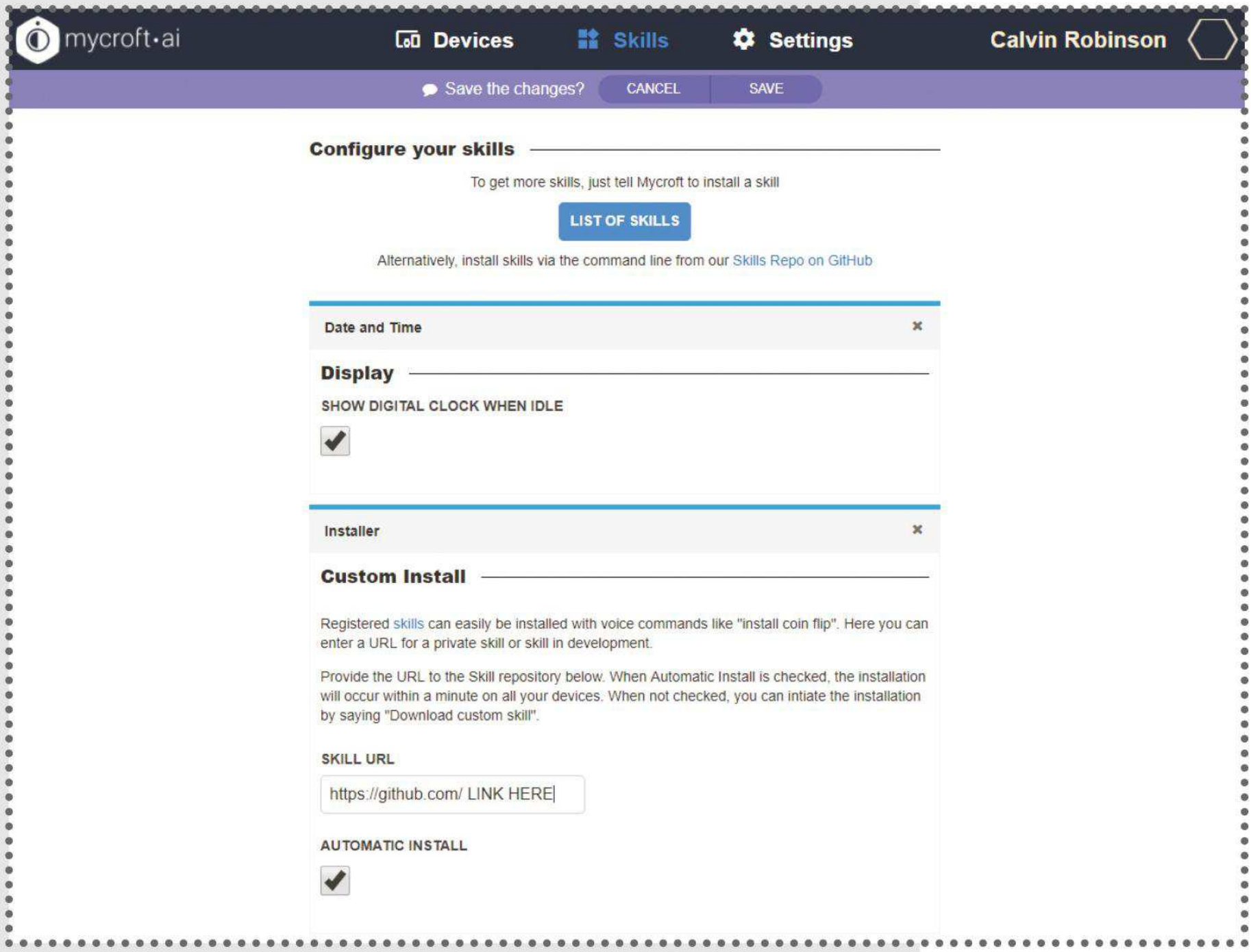
Hey Butler, what time is it?
Hey Butler, set an alarm for X am.
Hey Butler, record this.
Hey Butler, what is [insert search term].
Hey Butler, tell me a joke.
Hey Butler, go to sleep.
Hey Butler, read the news.

Hey Butler, set a reminder.
Hey Butler, do I have any reminders?
Hey Butler, increase/decrease volume
Hey Butler, what's the weather like?

You can also skip our earlier step of using nmap or arp by asking "Hey Butler, what is my IP address?".

08 Adding Skills

Of course, the default abilities are all well and good, but surely where an open source program comes into its own is with customisation. Mycroft/Picroft is no different in this regard, with a whole range of different voice abilities available. These seem to have been coined 'Skills' –



we can thank Amazon for that.

Back at Mycroft.ai, it's time to explore the Skills menu. There's an option to paste a GitHub URL to install a Skill, which is quite useful, but Mycroft does also recognise "install [name of Skill]" as a command. You'll see a link to a list of community-developed Skills, where you can also find the names and command needed to install them. "install YouTube" adds a simple YouTube streaming Skill, for example.

09 Play music

The only officially supported music-playing app seems to be mopidy, which we had great difficulty in getting working. Hours of fiddling with dependencies and an extended deadline later, we still had no luck. However, we did find spotify-skill in the GitHub repository, which works a treat.

Simply by copying the GitHub URL (**https://github.com/forslund/spotify-skill**) into the 'Skill URL' box on Mycroft.ai and ticking 'Automatic Install', moments later we had a new menu option to input our Spotify details. Then "Hey Mycroft, Play Spotify" loads up our most recent playlist. The only problem was that we couldn't figure out a way to stream directly to the Mycroft; spotify-skill only streams music to another Spotify Connect device. It's only speculation on our part, but we assume this is something to do with licensing restrictions for 'official' Spotify devices.

10 Play podcasts and radio

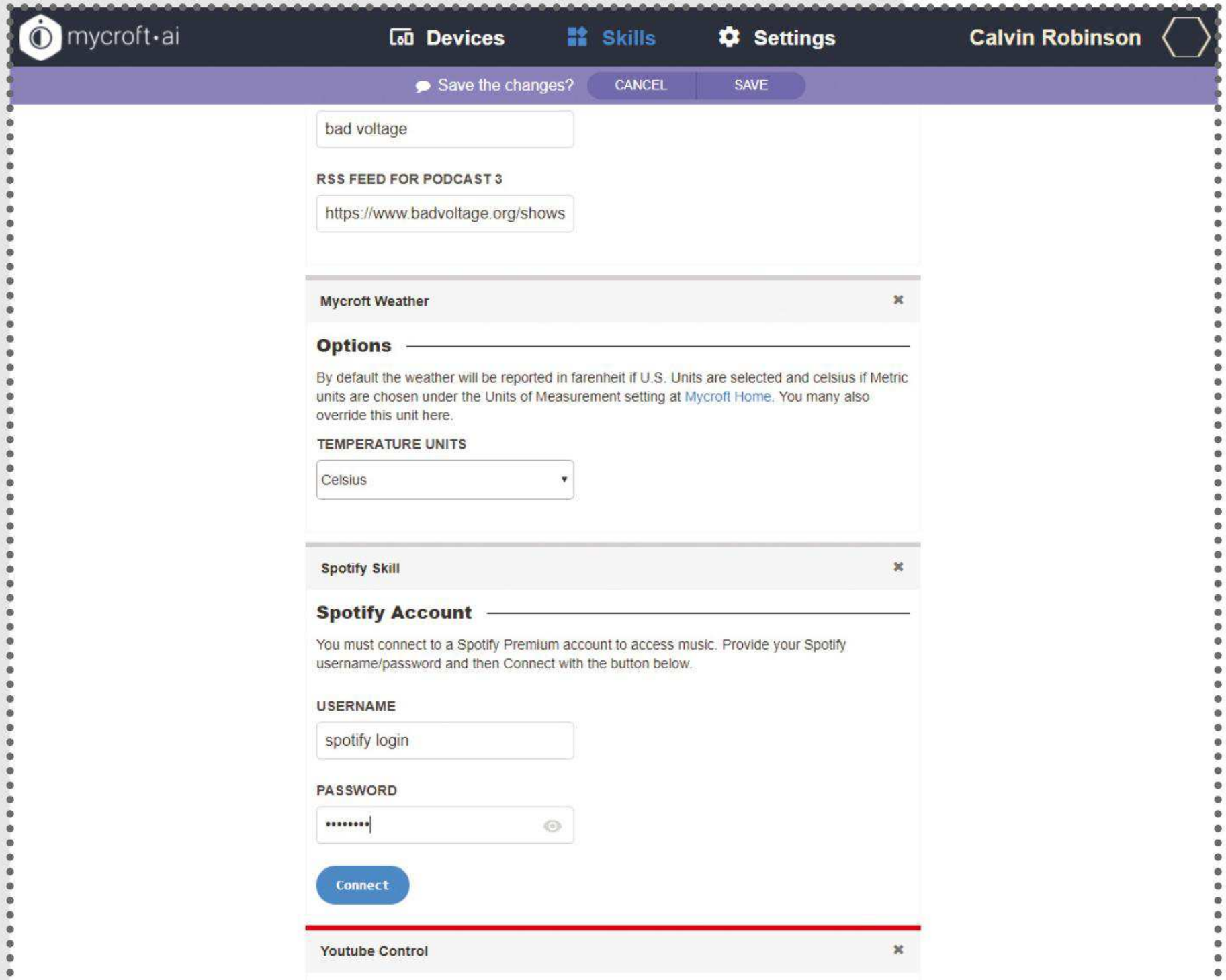
Thankfully, it's much easier to stream podcasts than it is stream music. A quick "install podcast skill" install the necessary Skill, and you'll then have options on Mycroft.ai for your three favourite podcast feeds. Paste in the RSS details and you're good to go. "Hey Mycroft, play x podcast"

RTFM...

At the moment Mycroft is available in several flavours. The version we're looking at here, technically known as Picroft, consists of the free software only – you'll need to add your own Raspberry Pi to run it, plus speakers and a mic.

If you prefer an off-the-shelf version, you can opt for the Mycroft Mark I ($180), a standalone hardware device which is equally 'hackable' in terms of adding abilities or changing code.

Finally, there's Mycroft for Linux, which you need to install using either a shell script or a standalone installer. Mycroft AI describes this as "strictly for geeks".



should then do the trick.

We didn't have as much luck with the Internet Radio Skill, though. Requesting any Internet Radio stations threw up PHP errors, which are visible in the Picroft command line interface and log viewer. It seems as if Skills are very hit-and-miss at the moment. There is a 'status' column for each one on the community page which is meant to indicate its readiness, but we found the results to be inconsistent.

11 Replacing commercial voice assistants

While the Picroft has been a fun experiment to sink (way too many) hours into, do we think it's ready for prime

time? In a word: no. While the core experience may be fine, it's extremely limited and the Skills are not yet up to scratch. In our experience, they're just not very likely to work, even after hours of fiddling.

If you're looking for a new hobby and don't mind putting a few days into this, you'll get some enjoyment out of it. However, if you're looking for a new voice assistant to read you the news, wake you up and play your favourite music or radio station, we're still forced to recommend one of the commercial units. Having said that, Mycroft Mark II is available to reserve on IndiGogo right now too.

12 Testing

It may be that Mycroft's voice recognition isn't up to scratch, or it may be that the microphone we used for testing was cheap and useless, but constantly issuing commands via voice during the testing process proved to be tiring. Fortunately, Mycroft supports text-based commands, too.

If you SSH into your Picroft you can type text commands directly into the command-line interface. If you exit out of the CLI, there are a number of command prompts available:

mycroft-cli-client A command line client, useful for debugging

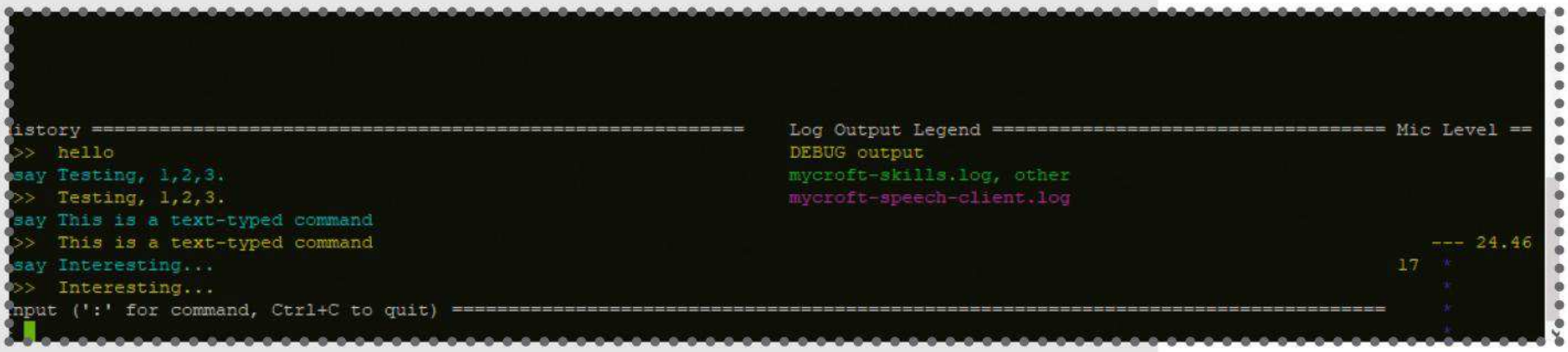
msm Mycroft Skills Manager, to install new Skills

say_to_mycroft Use one-shot commands via the command line

speak Say something to the user

test_microphone Record a sample and then play it back to test your microphone.

"If you're looking for a new hobby and don't mind putting a few days into this, you'll get some enjoyment out of it"



13 Becoming a supporter

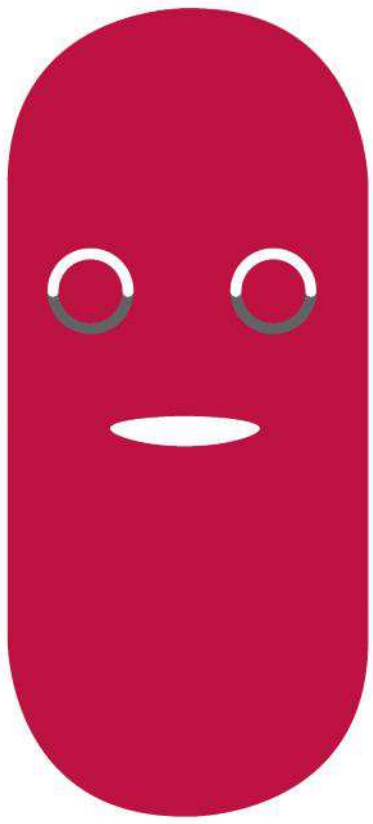
Mycroft offers an optional subscription service, at $1.99 per month or $19.99 for a year. While the primary purpose of these subscriptions is to support the development team, there are exclusive updates which are made available only to subscribers.

As of May 2017 there's a new female voice available only to supporters. There was also a Group Hangout session with John Montgomery himself in April. The missions statement reads:

"It's hard to overstate how much I value your support Calvin Robinson. It allows my team to make me grow, and become better, faster, stronger. Your contribution takes us all closer to the ultimate goal of creating a general purpose artificial intelligence, which is open for everyone."

14 So, is it worth it?

There are lots of pros and cons to a setup like this. There's the freedom of being able to create your own Skills, or to find them in the brilliant online community. There are the benefits of being able to jump into the code and have a play-around, or just to check that your data really isn't going anywhere. But you have to balance that against the inability to find a working Skill when, for example, you just want to stream some music. When we did manage to get a stream working, Mycroft would talk all over the audio stream, with false positives of the wake word being picked up.



If you're a hobbyist looking for a new project to sink your teeth into, Mycroft might be right up your street. If you just want a device that you can say "Play the Beatles" to, without getting out of bed, this might not be the right setup for you right now. That's not to say it won't ever be, with the community and Skills growing at a rapid pace – and with the very promising Mark II version on the way, who's to say what Mycroft might be in a years' time? At the moment, though, it's lacking commercial viability.

Below Mycroft Mark I comes with speakers and mic built-in

Kodak Pi Notification Lamp

Another mashup of retro and modern tech, this voice-activated Kodak lamp uses a Pi Zero W and Unicorn HAT

This month, we've highlighted a new project from Martin Mander, which is a little less complicated but no less classy than his 1986 Google Pi Intercom. This time another retro sale snagged Martin a 1930s Kodak table viewer that he's adapted to display notifications and alerts using a range of LED colours on a Unicorn pHAT (from Pimoroni, see https://shop.pimoroni.com/products/unicorn-phat). The pHAT has a programmable matrix of bright LEDs and this is controlled by a Raspberry Pi Zero W, which checks for incoming Gmail instructions using a simple Python script.

Where did this idea come from?

I picked up this Kodak slide/negative viewer at a sale recently for £10. Normally I do a bit of research before tearing a piece apart, just for curiosity and to get a feel for its original function, but I didn't get very far! Having trawled Google images and old photography catalogues, the only similar model I came across was an eBay listing, asking around £600 for the viewer. I have no idea if that's a fair price or not but decided for this project that I'd attempt a non-destructive conversion. This turned out to be a smart plan as the viewer is made exclusively of glass and metal, so any hacking around would have risked ruining an irreplaceable component. I decided to stay fairly true to its original function and replace the light source with a modern LED array, hoping this would be bright enough to glow nicely and project at least some light through the viewer lens onto a wall.

Martin Mander works as an analyst in Intelligence and Analytics at Norfolk County Council and has a passion for combining new technology with vintage design.

That rainbow lighting effect from the lens and the Unicorn HAT is fantastic [see https://youtu.be/B_vkmmcb0_M]. Have you used the HAT for many other projects?

I've experimented with the Pimoroni Unicorn pHAT before, in the ill-fated sequel to my Rabbit Pi project, so it immediately sprung to mind when I started thinking about LED options. The LEDs are bright to the extent of having a health warning and are easily programmed using Python, so this was the ideal choice. I wanted the pHAT to sit where you would normally put a slide, but unfortunately this gap was only about 7mm.

I found that the Unicorn pHAT only needs to be connected to three of the Pi's GPIO pins (5V, GND and GPIO 18) and this was a real life-saver – it meant I could solder in single right-angle headers to just those pins on the board and keep the profile nice and slim. I dismantled the metal slide aperture – which had teeny screws – and lightly glued the pHAT to the back of it, so that as many LEDs as possible would shine through the lens.

You mentioned an 'ill-fated' sequel to your cute Rabbit Pi project. What happened?

It was a cool project, connected to Alexa like the original but using relays to control the ear motors and with the Unicorn pHAT making the entire rabbit glow instead of single LEDs – what I didn't factor in is that when you use the pHAT it interferes with analogue audio playback, which really complicated the Alexa integration. I experimented with HDMI and USB audio

THE PROJECT ESSENTIALS

- Raspberry Pi Zero W with 40-pin header soldered in
- microSD card
- Pimoroni Unicorn pHAT
- Right-angle header pins for the pHAT (8 used)
- 3× 20cm female to female jumper leads
- Heat shrink to disguise the three leads
- Plastic box to chop up for the bracket
- 1× M10 bolt to secure the bracket to the cylinder door
- 1× 1930s Kodak Table Viewer
- HDMI and USB adaptors (for access to monitor and keyboard)

adaptors but after a while it stopped being fun so I moved on. The rabbit may rise again when my Google AIY pre-order comes through, as this makes the motor control and audio side much easier to achieve.

How did you add the interactive aspect of the project?

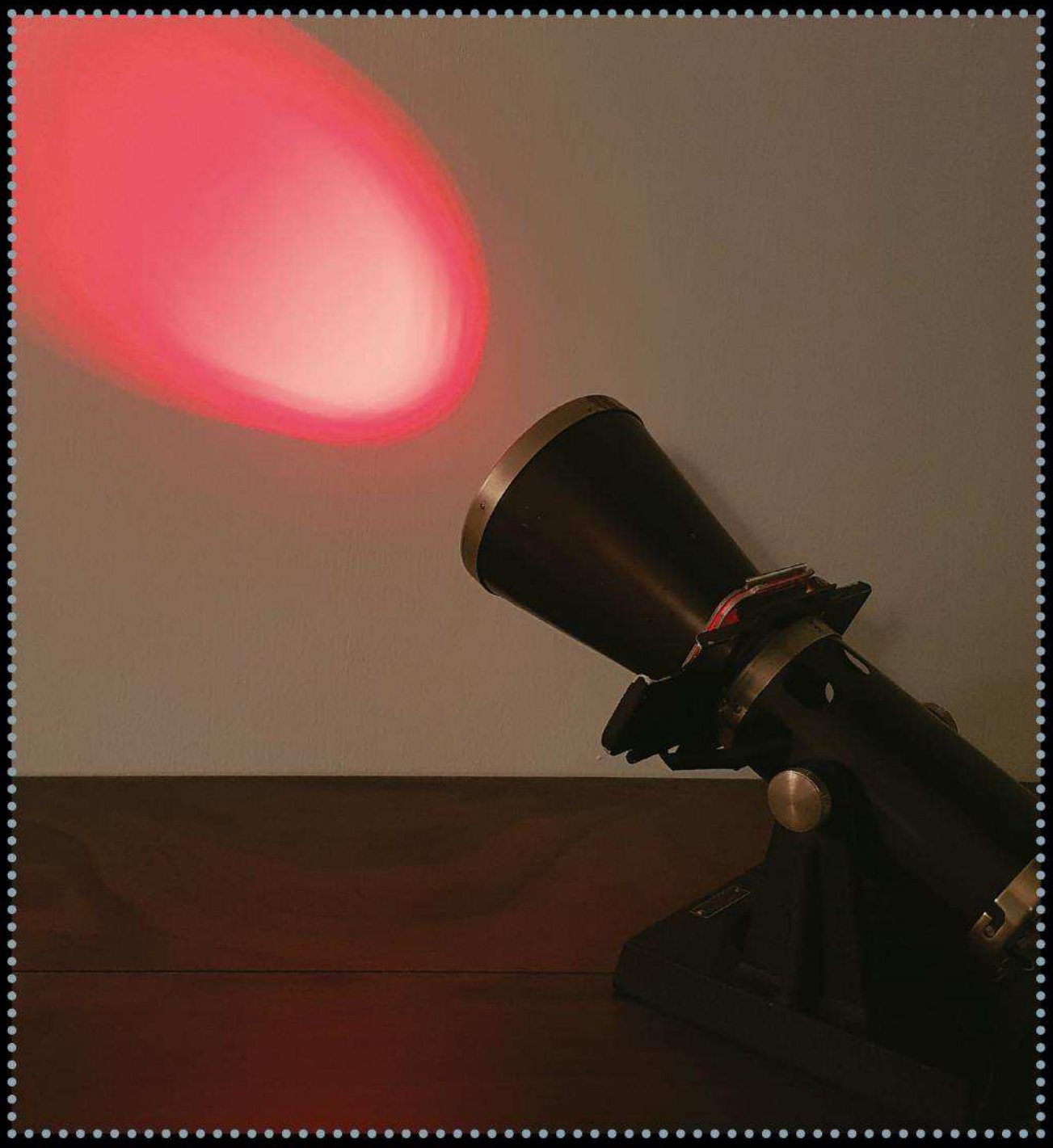
I wanted this to be a connected, interactive lamp, so I set about pulling together some code to get the Pi Zero online. I reused code from my Talking Radio project [http://bit.ly/TalkingRadio] as a starting point, which uses a Python script to check incoming Gmail messages for a specific character string. After installing the necessary code for the Unicorn pHAT, I adapted some examples so that the Pi would light up the pHAT

Below Martin says he especially enjoyed the extra challenge of "not irreversibly altering the original piece. I just love that big lens on the front, and the way the whole thing can be angled and tilts open." Although quite heavy, it's also quite portable and only needs a single USB plug for power; so, with all its useful scripts, he's found it handy that he can move it around the house

in different colours depending on the subject text of the Gmail message it received; for example, if the word 'green' was included, it would light the LEDs green for 30 seconds. Once the script was working properly, I saved it into the /home/pi folder and set it to auto-run on boot by adding the line @sudo python /home/pi/kodak.py to the end of the .config/lxsession/LXDE-pi/autostart file.

What part of the project were you most pleased with?

I'm really pleased with how the plastic tray holds the



Like it?

Another clever voice-activated project from Martin is the Talking Radio (http://bit.ly/TalkingRadio). This revives a 1940s DeWald radio he bought for £3, with the help of a Pi Zero, a Blinkt! LED strip and a pair of PC speakers to enable it to read out notifications from a host of internet-connected services.

Left As well as his basic email/flash demo, Martin uses IFTTT to create some more practical scripts, such as making the lamp flash a colour if his phone battery (using Android Battery Service) gets below 15%, or flash different colours when an outside camera detects motion. He's also linked it up to the Weather Underground so that it lights up blue if it's about to rain

Pi securely in the back part of the viewer. It's satisfying and practical to be able to slide out the Pi to connect up a monitor and keyboard if I need to. The back part of the viewer originally held the bulb and its holder, and was conveniently big enough for a Pi Zero, as long as it had no extras attached (hence, I chose the W version with Wi-Fi built in). The back of the cylinder locked in place with just a twist, so I decided to keep this feature and attach the Pi in place of the bulb holder.

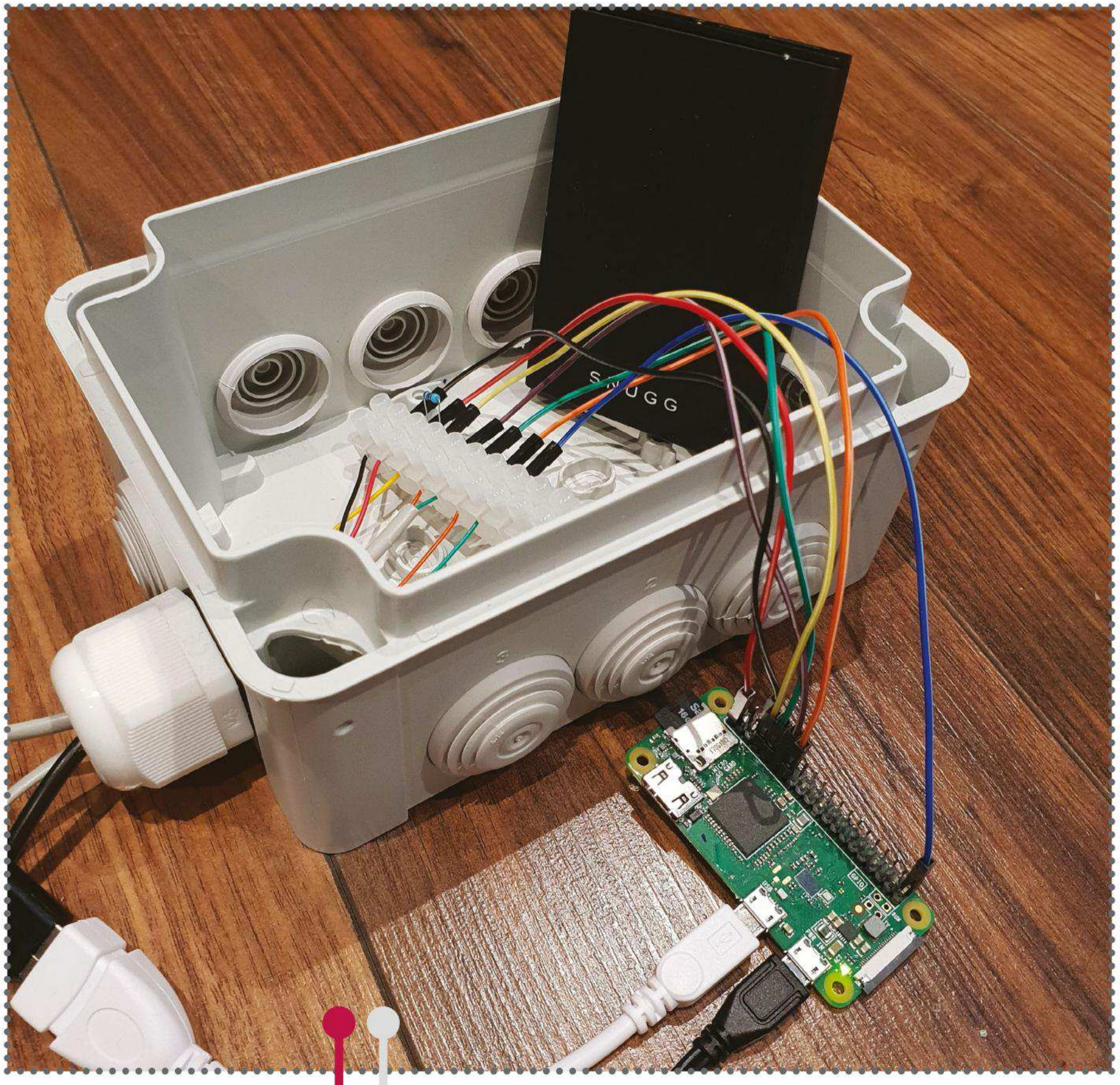
After a few experiments, I cut out a section from a translucent plastic box with a rotary tool, drilling holes so that it could be bolted to the 'back door' and the Pi.

Further reading

If you want to have a go at a similar project. Martin's code is at http://bit.ly/GHKodakPi and you can follow his full walkthrough at http://bit.ly/Kodak-Pi.

Build a weather station

You don't need to train in meteorology to understand the weather – just a Raspberry Pi and a kit



Maker Life's tagline is 'coding for kids'; it produces pre-packaged development kits for parents, teachers and children with an aim of making coding fun and easy at low cost. We're going to take a look at its most comprehensive kit, the Weather Station, and see just how much one can learn about the world around us from a Raspberry Pi Zero W and a bunch of sensors.

This kit is a bit of a doer-upper. We'll need to spend some time building the weather station and putting the sensors together before we can get down and dirty with any Python code. There are no soldering requirements, but the kit is moderately difficult to assemble.

It seems important to Maker Life that its kits are as accessible as possible, and are therefore designed to appeal to girls as much as boys. We're going to put these claims to the test and recruit the assistance of some GCSE Computer Science students for this one. All assembly and coding in this tutorial has also been verified and tested by a group of female teenage coders.

01 Checking for parts

Before you partake on any journey into physical computing, it's always important to check the parts first. Halfway through assembly we thought we'd lost two essential pieces of the puzzle; it was only moments before we were resigned to order replacement parts that we discovered the white boxes camouflaged at the bottom of the larger box.

This particular kit should contain: Raspberry Pi Zero W; microSD card; power supply; rain gauge; temperature sensor (DS18B20); anemometer; waterproof

container; mast; gland; Micro-USB to Micro-USB extension cable; USB hub; HDMI cable; HDMI to Mini HDMI adaptor; USB to Micro-USB adaptor; terminal block; 10x male-to-female connectors; 4.7K Ohm resistor; and a guide.

02 Waterproofing

Obviously, given that our weather station will be used outside, it's essential that it is somewhat waterproofed. There's a water-resistant junction box included in the kit; you'll need to decide which side will be at the top and bottom, and then remove a rubber grommet from the bottom of the unit (usually the shortest side). Replace that grommet with the included gland, through which we'll thread all our wires – it needs to be on the bottom so as to any avoid rain seeping in when we're done.

Thread through all the cables you'll be connecting to the Raspberry Pi from the sensors, including the wind sensor, thermometer and rain gauge. These sensors

Other versions of Mycroft

After writing out our somewhat buggy first draft of code, we misplaced our instructions manual. There doesn't seem to be a digital version for this particular kit anywhere online, so we were stuck figuring things out for ourselves. While that was a fun adventure, it might not be your idea of a good time – so whatever you do, don't misplace that manual!

should have exposed wiring at the end; if they don't, you'll need to pull the end off (ours had an odd Ethernet/phone-style terminus) and strip the wires yourself ready for the next few steps.

03 Terminal circuitry

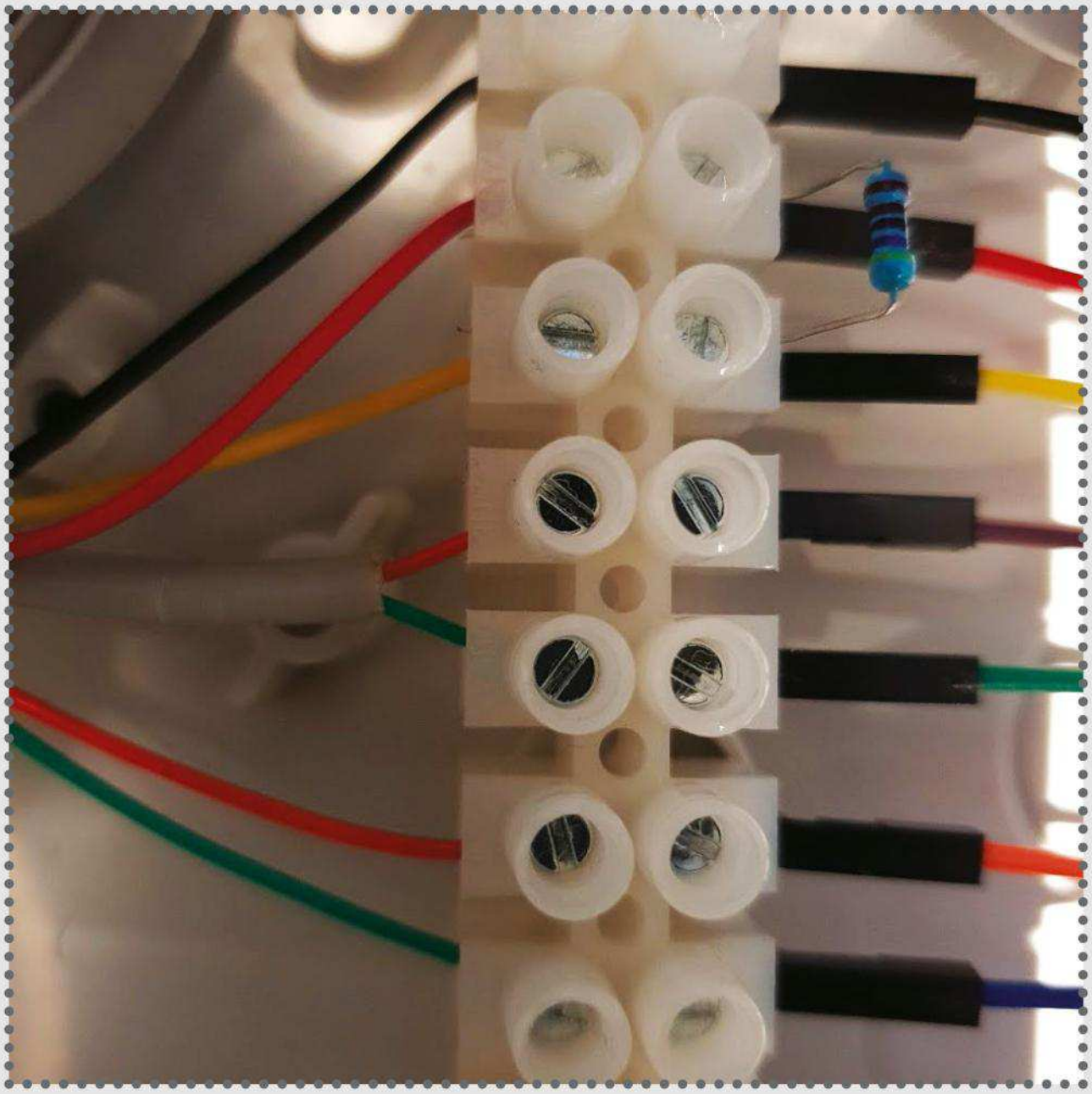
The included terminal block is used to transition the stripped wire-ends to the male-to-female jumper cables and connectors that can be plugged into our Pi's GPIO headers.

You can use any order you like, but to make life easier we'll do them in order of thermometer, wind sensor, rain gauge. Unscrew all the terminal screws (not all the way out, though), pop the three copper cables from the thermometer into the terminal block and secure

them in place by screwing them down. From left to right, that should be black, red and yellow. Next, we'll do the orange and green wires from the wind sensor (again, left to right respectively) and then orange and green from the rain gauge. Once these are all secured into place we can work on the other side.

04 Terminal block to Pi

This is where things get a little tricky. On the reverse side of the terminal block we want to connect the male side of the male-to-female connectors in the following order from top to bottom, assuming the black cable from the thermometer is at the top: black, red, yellow, purple, green, and orange. However, between red and yellow you'll need to also shove in the ends of the 4.7K Ohm



resistor. This is to prevent too much power going through the thermometer. The resister should fit snuggly beside the male pin. Secure all the screws as before. You'll notice there are five empty slots on the terminal block, which is normal; you could use these for additional sensors.

05 Connecting the Pi

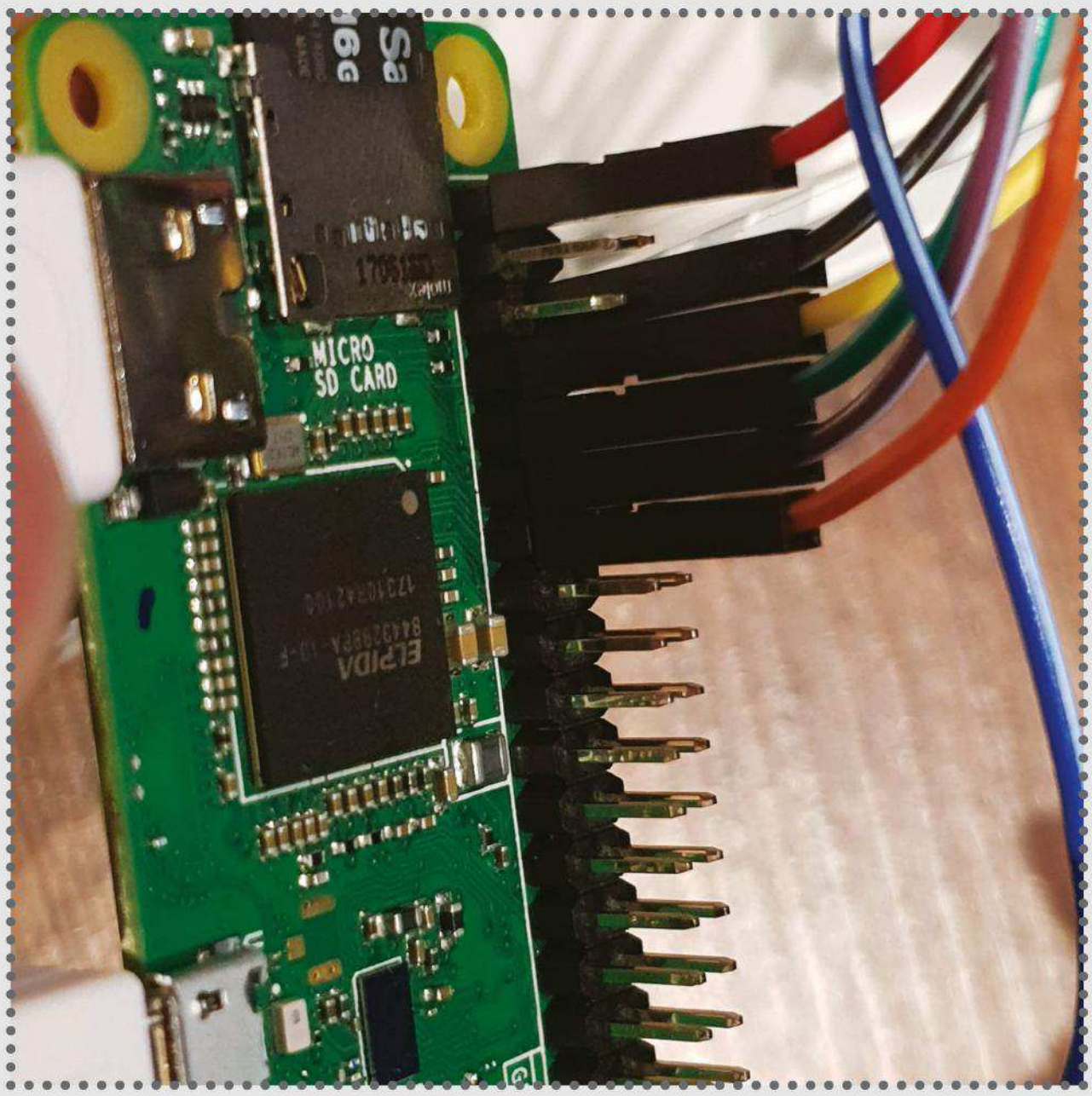
Using the Raspberry Pi Zero W's GPIO chart, connect the female end of the male-to-female jumper cables/ connectors to the following pins: black, slot 6 (Ground); yellow, pin 7 (BCM 4); green, pin 9 (Ground); purple, pin 11 (BCM 17); orange, pin 13 (BCM 27); and the red cable on pin 1 (3v3 Power). Blue goes all the way down the bottom on pin 40 (BCM 21). Pop the power extension cable through the gland, too. That should be all the external cables and wiring set up. Now you may want to use the USB hub and HDMI adaptor to connect a mouse, keyboard and display monitor temporarily, while we code on the device. These can be removed once everything is set up and before we close the waterproof container.

06 Operating system selection

Put in the SD card, plug in the power and we're good to go. Be sure to check that the power and USB hub are in the right way around – they're labelled on the Pi's board as left for USB, right for power – before turning on the power socket.

We're almost ready to get coding. Before we do, we'll need to download and install an operating system. NOOBS is installed by default on the microSD card, so provided you have a Wi-Fi connection available, you can download an OS from the selection. We recommend

going with the top option, Raspbian. If you don't have a Wi-Fi connection you'll need a USB Ethernet adaptor and cable to get set up.

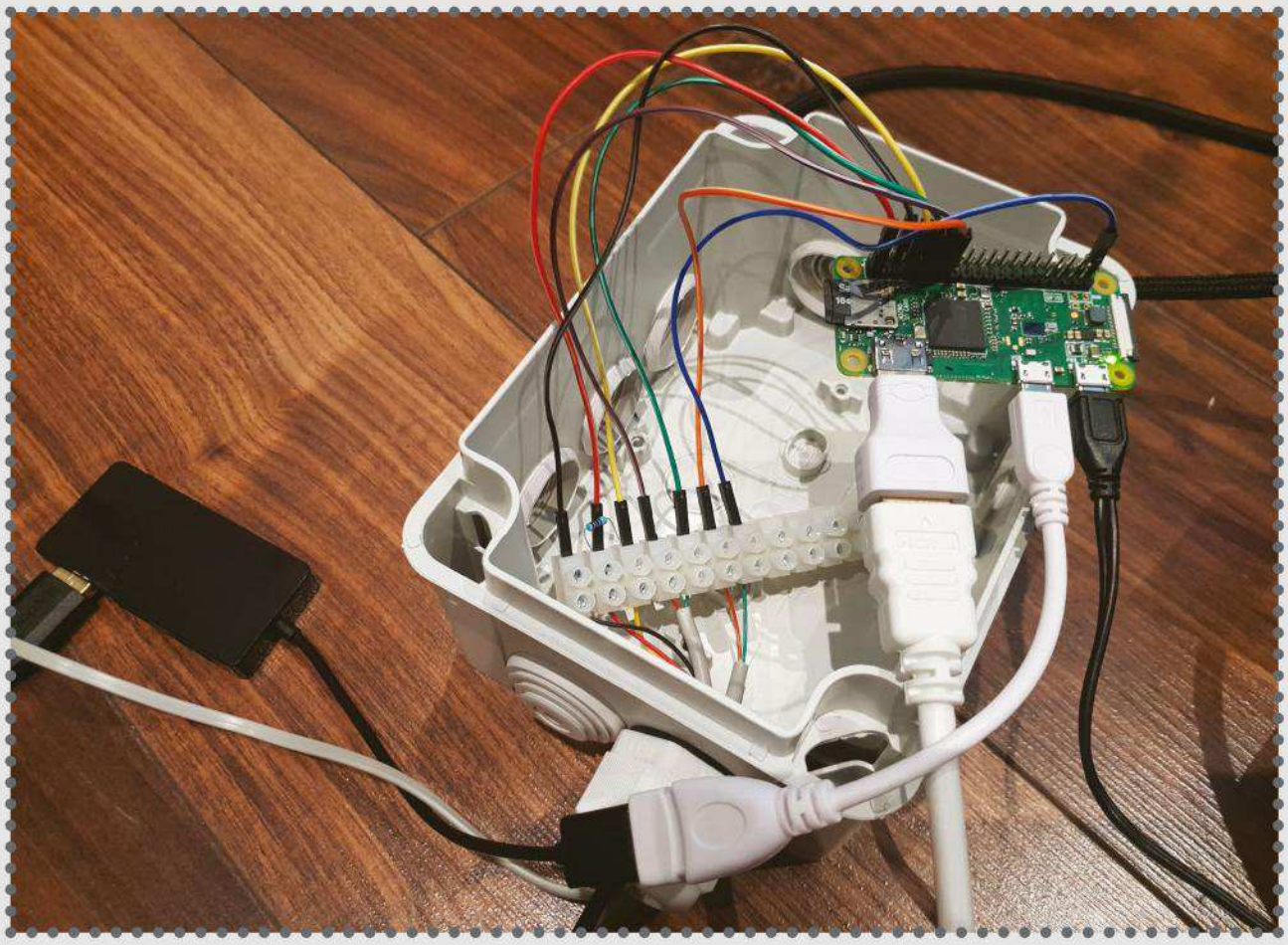


07 Pre-programming Python setup

We discovered – eventually – that a few setup steps were necessary to get any code working with the sensors in our weather station. You'll need to run the following commands:

```
sudo apt-get update
sudo apt-get install python3-setuptools
python3-w1thermsensor python3-gpiozero -y
```

This will also install python3-click. You'll need all of these before you can use the Python 3 IDLE (with the emphasis on 3!) to interface with the thermometer in particular. It took us a few hours of dependency hell to work out this particular puzzle; luckily, thanks to our selfless sacrifices, you don't have to.



08 Pre-programming Pi setup

There's one more step before we can tuck into some code. Run sudo nano /etc/boot and add the following line at the end:

```
dtoverlay=w1-gpio,gpiopin=4
```

Where gpiopin=4 is the GPIO pin of your DS18B20 thermometer (not the ground or power pin, but the one being used by the Pi to send and receive data signals). In our example earlier we used BCM4 aka GPIO 4, which is pin 7. Trust us on this one…

09 Start coding

Now that we've got the hardware set up and all the pre-configuration out of the way, we can begin to program our weather station. First let's import all the modules we'll need:

```
import threading, math
from time import sleep
from gpiozero import DigitalInputDevice
from w1thermsensor import W1ThermSensor
```

10 Global constants

Next, we'll set up some variables instead of having to remember all manner of numbers. Having global constants for things like the bucket size and how many centimetres in a kilometre will come in handy.

```
wind_count = 0
bucket_count = 0
radius_cm = 9.0
interval = 5
ADJUSTMENT = 1.18
CM_IN_A_KM = 100000.0
SECS_IN_AN_HOUR = 3600
BUCKET_SIZE = 0.2794
```

11 Objects

Now, we just need to create instances of our sensors, before we set up functions and activate them:

```
    sensor = W1ThermSensor()
    wind_speed_sensor =
DigitalInputDevice(17, pull_up=True)
    rain_sensor = DigitalInputDevice (27,
pull_up=True)
```

12 Setting up a rain function

Using our global constants we can create a function to define rainfall based on how much rain is in the bucket relevant to its size. We'll then print a message letting the user know:

```
    def rain():
    global bucket_count
    bucket_count = bucket_count + 1
    print("Rainfall is currently
"+str(round(bucket_count * BUCKET_
SIZE,2))+"mm\n")
```

13 Activating a sensor

We'll gather data from the rain gauge to detect when it's raining (indicated by the sensor being activated), and run the previous rain function to monitor the amount of rain and report it to the user.

```
    raindata = threading.Thread(name='rain',
target=rain)
    raindata.start()
```

```
rain_sensor.when_activated = rain
```

14 Functions and sensors

We'll do a very similar thing with wind and temperature; you can find the full code on the coverdisc in a file called weatherstation.py.

Once we have a function and instance of each sensor set up, we can run the code and see live read-outs any time one of the sensors is activated, by detecting a change in the relevant area.

Pour some water into the rain gauge, spin the wind sensor, or hold the thermometer in your hand, and assuming everything is wired up correctly you'll see data being printed to the screen in real time.

15 Pricing up and buying your own parts

Maker Life's kit costs £85. As an experiment, we priced up parts from AliExpress (www.aliexpress.com) and weren't able to cut much off the cost, even by buying in bulk, once you factor in all the shipping costs. In fact, the pricing is spot-on and Maker Life doesn't seem to have much of a profit margin on this kit.

If you did want to put one together yourself, you'd need to order the following parts:

- Wind Sensor (anemometer) ~£30
- DS18B20 ~£1
- Resistor ~£0.80 (for 100)
- Raspberry Pi Zero W ~£15
- Schneider Electric ENN05007 EP 55 junction box ~£5.80
- Gland ~£1

Teamwork makes the dream work

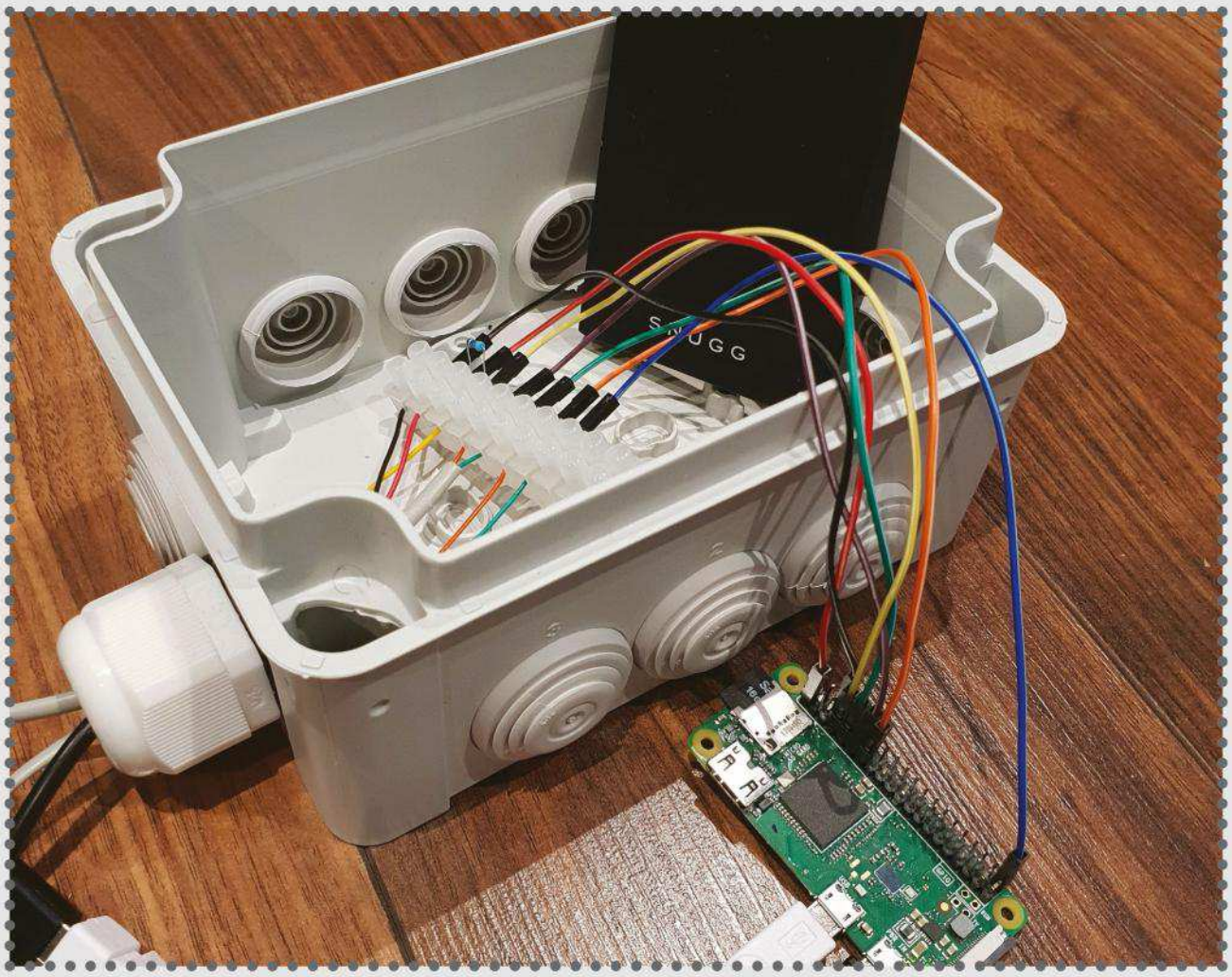
This is a great project for collaborative working between an engineer interested in physical computing and a coder who wants to put their Python programming skills to the test. This kit is very much in-line with the Raspberry Pi Foundation's philosophy of supporting a new generation of 'Digital Makers'.

- USB hub ~£10
- Terminal Block ~£1
- Rain gauge/meter ~£10

16 Next steps

To make this station truly portable you could add a power bank to the junction box, to power the Pi. Provided there's a wireless connection, you could then take the weather station anywhere and simply connect remotely via SSH or VNC to monitor the readings. Or you could attach a touchscreen or LCD to display readings.

Some additional sensors you may want to include: motion sensors, humidity sensors, a barometer for air-pressure, moisture sensor, capacitive ground moisture sensor, MQ gas sensor. We'd love to hear about how you're adapting and improving your weather station, so please tweet us photos and suggestions to @LinuxUserMag.

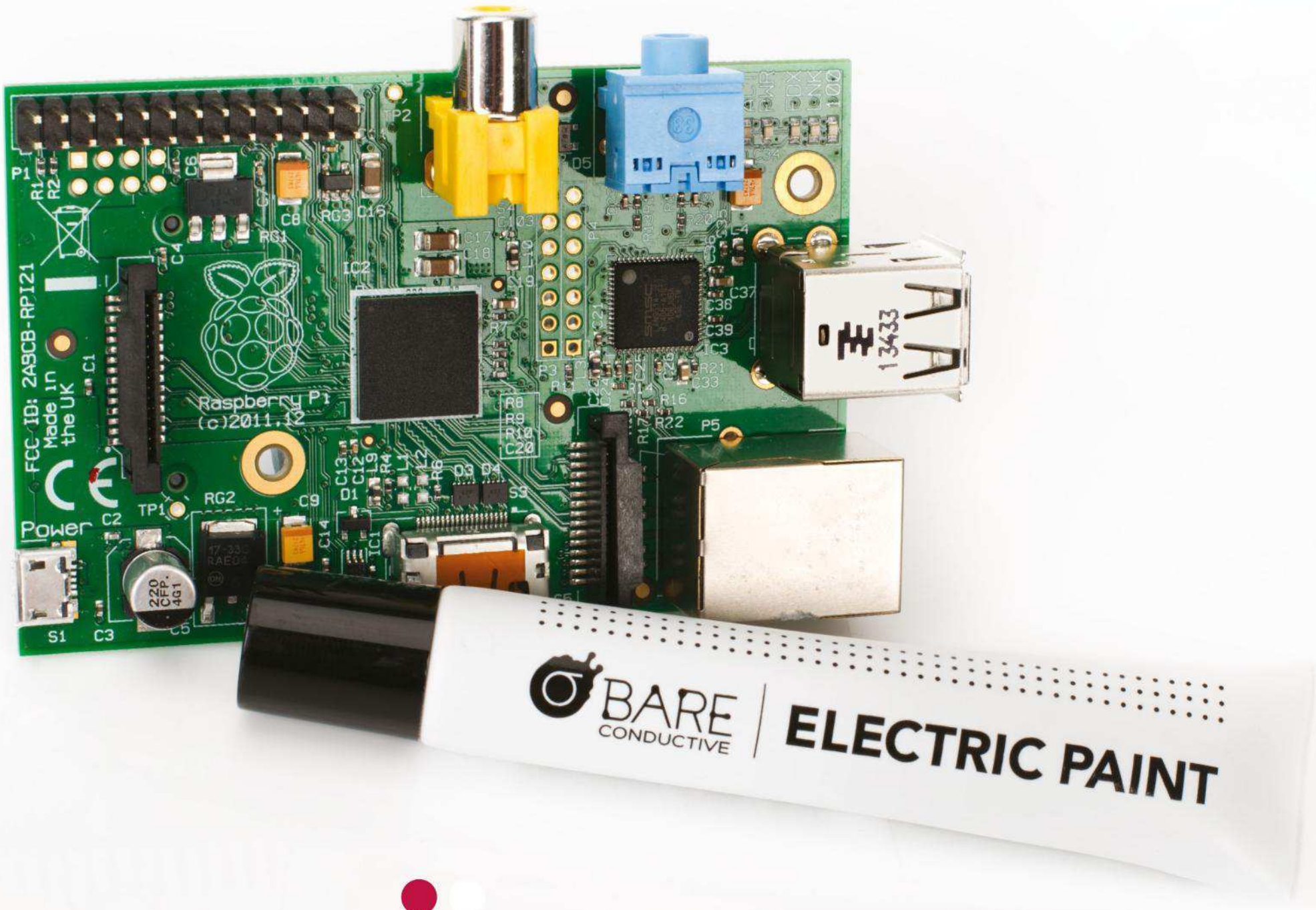


Take it to the next level

If you've ever wanted to tinker around with the GPIO pins on your Raspberry Pi, create your own circuit and write programming code for it, this is the project for you. It wouldn't be very difficult to take this kit to the next level by introducing your own sensors for monitoring ambient light, noise and even pollution levels. It'd only take the addition of a microphone, LDR sensor and air-quality HAT, none of which are very expensive, and there's plenty of room on the Pi's GPIO board for additional circuits.

Draw circuits with paint

Assembling circuits has never been so easy with the joys of conductive paint, enabling you to bring together art and electronics in a whole new way



Playing with electronics and physical computing is a very rewarding task. For a beginner though, the mess of wires and components can become very confusing quite quickly and things like soldering can be a safety concern when children are involved. Bare Conductive has taken the joy of electronics and made it far safer, easier and more versatile with their conductive paint. You can literally draw wires on paper with a paintbrush, use it for cold-soldering or a conductive adhesive and much, much more. There are not a great deal of boundaries to what you can do. Pair this paint with a microcontroller board and you could be creating interactive art, clothing and projects in no time.

THE PROJECT ESSENTIALS

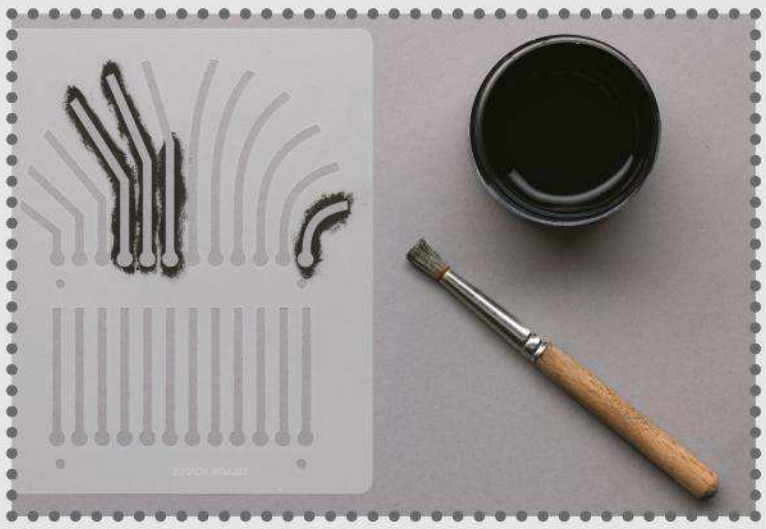
- **Bare Conductive paint** (pen or tub)
- **Male to female jumper wires**
- **An assortment of LEDs, switches and resistors** (optional)

01 Get your tools

Paint and a paintbrush aren't the first items that come to mind when you think about electronics, so you may be wondering where to get them from. Bare Conductive stock the paint and a selection of components in their shop **(bareconductive.com/shop)** but you will need to go somewhere else for art supplies. We would recommend trying a high street craft shop such as Hobbycraft **(hobbycraft.co.uk)** or a local independent.

02 Pick your platform

The great thing about Bare Conductive paint is that, when dry, it works just like normal wiring! That means you can use it with any of your favourite microcontrollers like the Bare Conductive Touch Board, a Raspberry Pi or Adafruit's wearable FLORA platform. Or you can just use some small pin batteries and flashing LEDs for a standalone system.



03 Start to paint

You can paint Bare Conductive paint onto pretty much any surface – paper, fabric, walls, clothing, wood, plastic and much more. For really accurate shapes and results, the best idea is to create or purchase a stencil (paper stencils are easiest to make at home but use vinyl for the best edge finish).

04 Connect it up

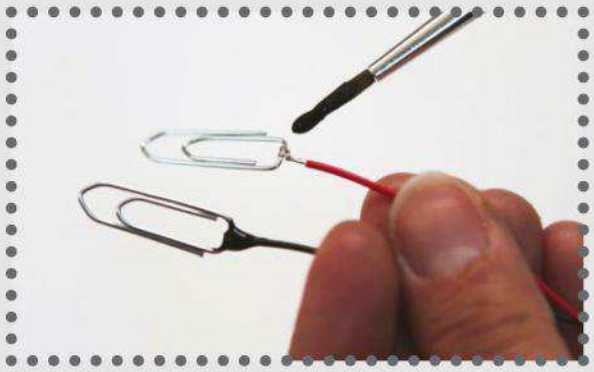
There are plenty of ways to connect to the conductive paint (from battery packs for example) no matter what surface it's on, because once it is dry it acts just like an uninsulated wire. Therefore you can use wires glued on with the paint, paper clips, bulldog clips, alligator clips or even sewn-in conductive snaps for wearable projects.

05 Make repairs

The conductive paint is thick and when it's dry it becomes quite strong. These means you can use it to cold solder things together and repair any breakages. In other words, you could glue components into a circuit board or glue wires together and they would still function electrically.

06 Clean up

A lot of you are probably thinking that something as cool as conductive paint is going to be nasty stuff. Actually



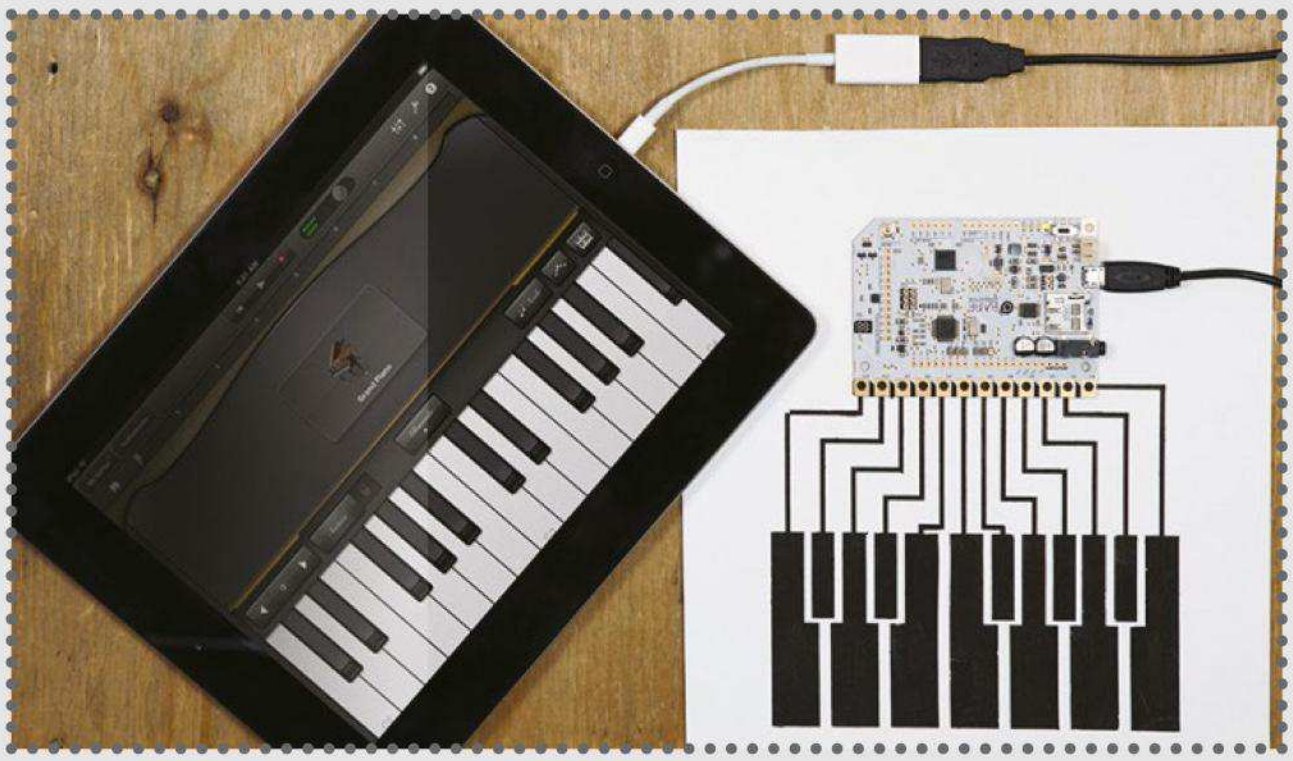
Bare Conductive paint is non-toxic, water-based and water-soluble, and can therefore be cleaned easily with soap and water.

07 Make it waterproof

This paint only comes in black and is not waterproof. However, the great thing is that you can use it underneath or alongside any regular paints, varnishes and waterproofing sprays in order to act as insulation – or just to add some colour into your designs.

08 Touch and sound

Bare Conductive paint can also be used as a capacitive surface, meaning you can use it for touch, gesture or proximity controls when it is paired with a suitable control board. Bare Conductive make their own called the Touch Board which has everything you need to start experimenting with touch and sound. It can even act as a MIDI controller, an interface or an instrument.



Using Numba to speed things up

This issue, we look at how to use Numba with your Raspberry Pi Python code to speed everything up and squeeze out extra performance

One area that is always an issue with Python code is performance. Much of this is due to using programming techniques from other languages that simply don't work the same way in Python. These types of issues are usually dealt with by rewriting your code in a more Pythonic form.

While this is perfectly adequate in most cases, people are always keen to squeeze every last bit of performance out of their code. In these cases, you have a few different options. Here, we'll look at one of them named Numba, which compiles your code using a JIT compiler based on additions to your code that enable you to fine-tune the JIT compiler.

This has traditionally been an issue for the Raspberry Pi, since it requires the LLVM compiler and this hasn't been easily available on ARM architectures – but the latest versions of Raspbian include a llvmlite package, allowing you to use Numba. In order to install it on your Raspberry Pi, use the following lines of code:

```
sudo apt install libblas-dev llvm
python3-pip python3-scipy
pip install llvmlite==0.15.0
pip install numba==0.30.1
pip install librosa
```

You do need to be careful of version numbers, as this requires patches that may not be part of the stable branches. Depending on your setup, you may want to do this inside a virtualenv. If so, run the following commands first:

```
virtualenv --system-site-packages
-p python3 env
source env/bin/activate
```

If you're developing on another box, you should probably use an Anaconda installation. If so, install Numba with the following command:

```
conda install numba
```

Numba does code optimisation based on decorators that you can add to your code. These decorators invoke the LLVM compiler to generate code tuned to your particular CPU architecture. The easiest way to use it is to perform the default 'lazy' compilation process on your defined functions, such as this example:

```
from numba import jit
@jit
def my_sum(x, y):
    return x+y
```

Why Python?

It's the official language of the Raspberry Pi. Read the docs at https://www.python.org/doc/

Lazy compilation means that Numba won't bother compiling a particular decorated function until the first time it's called. At that time, the input parameters are analysed for type and a specialised compiled version is generated. Because the compiled version depends on the input types, a new version gets generated when input parameters of different types get used. This preserves the naturally polymorphic nature of the Python language.

In many cases, however, you know what the datatypes are supposed to be for a particular function. In these cases, you can tell Numba what it should be expecting in terms of datatypes, so in our example:

```
@jit(int32(int32,int32))
def my_sum(x, y):
    return x+y
```

This tells Numba that it should generate code that takes int32 as input datatypes and that it should also return int32 datatypes. There are several options you can give to Numba to help get the fastest code possible. This greatly depends on what your code is actually doing and is very much of the 'Your mileage may vary' type. The first to look at is the nogil option. If your code is thread-safe and will not impact, nor be impacted by, other threads, you can explicitly tell Numba that it can give up any locks on the GIL (Global Interpreter Lock), like this:

```
@jit(nogil=True)
```

Normally, these compiled code objects only exist during

the runtime of your object. You can save the compilation step by adding cache=True to the jit decorator. This tells Numba to save any compiled objects to files in the file system so that they're available the next time you run your program.

By default, the code generated by Numba tries to be as optimised as possible, which means it doesn't use the Python C API. If Numba can't produce code in this mode (referred to as 'nopython'), it will fall back to generating code that does use the Python C API (referred to as 'object'). If you want to know when this happens, you can add the option nopython=True to the JIT decorator. This tells Numba to throw an error when it needs to fall back, rather than simply doing it silently. This option is necessary if you want to try the latest experimental feature, automatic parallelisation:

```
@jit(nopython=True, parallel=True)
```

This will analyse your function and see if it can be parallelised, as well as seeing if many other optimisations can be applied.

There will be cases where you want even finer control over what Numba does with particular functions. In these cases, use the @generated_jit decorator rather than the @jit decorator. The big difference is that you have more control over the datatypes used. For example, the above example looks like this:

```
from numba import generated_jit, types
@generated_jit(nopython=True)
def my_sum(x, y):
```

```
if isinstance(x, types.Float):
if isinstance(y, types.Float):
return x+y
```

This decorator also accepts the other compiler options, such as nopython and cache.

While the default operation of Numba works on the assumption that it will be used as a JIT compiler, this isn't the only way you can use it. You can also use the Ahead of Time (AOT) mode to compile the necessary functions before they are used. The big advantage of this comes into play when you want to distribute your program to other people. If you use the default JIT activity, anyone you share the code with will need to have Numba installed. Using the AOT functionality means they'll be able to run your optimised code without Numba.

In order to take advantage of this functionality, you need to import the CC portion of the Numba module. The biggest restriction with this method is that you need to define everything up front, as you would with a traditional programming language. A simple example would look like this:

```
from numba.pycc import CC
cc = CC('my_module')
@cc.export('multf', 'f8(f8, f8)')
@cc.export('multi', 'i4(i4, i4)')
def mult(a, b):
    return a * b
if __name__ == "__main__":
    cc.compile()
```

Here we have two versions of the multiplication function defined, one for integers and one for floats. When you run this code, Numba produces a compiled module, named my_module, that you can share. When they import the compiled my_module, users will have access to the Numba-compiled versions of these functions.

There are several other options to tune the behaviour of Numba. One example is the @jitclass function decorator. This defines a specification for a class so that Numba can compile a specific version for your particular use-case. The following code provides an example:

```
    import numpy as np
        from numba import jitclass
        from numba import int32, float32
        spec = [('value', int32),('array',
float32[:]),]
        @jitclass(spec)
        class Bag(object):
            def __init__(self, value):
                self.value = value
        self.array = np.zeros(value,
dtype=np.float32)
```

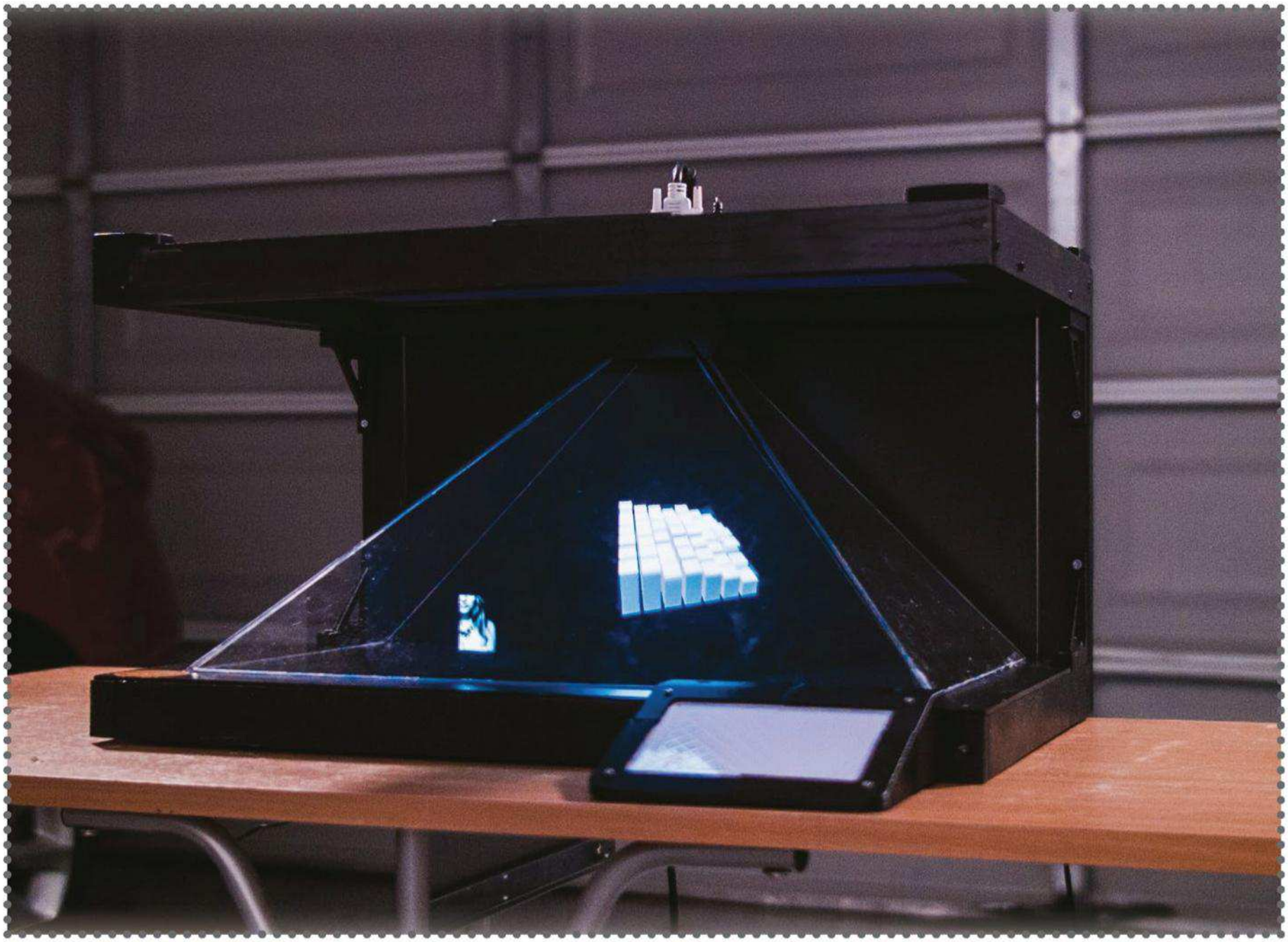
As you can see, you can optimise entire classes as well as individual functions.

Hopefully this short article has given you some ideas that you can use for your own projects. There are several ways to speed up your Python code, and while this is just one of them, it should give you a place to start.

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

"Turn your music into holograms"



Get this issue's source code at:

www.linuxuser.co.uk/raspicode